Amendments to the Claims

Please amend claims 1, 3, 5, 12, 14, 16. 23. 25. 27 & 29 and add new claims 36-39 as set forth below, and cancel claims 2, 10, 11. 13, 21. 22. 26, 34 & 35 without prejudice. In accordance with current amendment practice, all pending claims are reproduced below. The changes in the amended claims are shown by underlining (for added matter) and strikethrough/double brackets (for deleted matter).

1. (Currently Amended) A method of evaluating a software component, comprising:

deriving a conceptual layering scheme of the software component, the conceptual layering scheme dividing the software component into multiple software layers, the multiple software layers taking into account relationships that exist between software layers;

providing an abstraction matrix in mathematical abstract form that describes the multiple software layers of the software component and accounts for the relationships that exist between the software layers, the abstraction matrix further comprising at least one test case scenario for each software layer and mapped expected results ~~therefore~~ therefor;

testing the multiple software layers of the software component using the test ~~cases~~ case scenarios to generate test results; ~~and~~

evaluating the test results using the abstraction matrix, the evaluating including for each software layer automatically comparing a test case employed in the testing of the software layer to the at least one test case scenario of the abstraction matrix therefor, and if a match is found, automatically comparing the test result for that test case with the mapped expected result ~~therefore~~ for that test case scenario in the abstraction matrix; and

wherein evaluation of the software component is accomplished upon completion of evaluation of the test results.

2. (Canceled).

3. (Currently Amended) The method of claim 1, further comprising, prior to said testing, deriving at least some test cases <u>for each software layer</u> from the at least one test case scenario of the abstraction matrix.

4. (Original) The method of claim 1, wherein the software component comprises multiple states, and wherein the abstraction matrix comprises multiple test case scenarios, each test case scenario being associated with a different state of the software component.

5. (Currently Amended) The method of claim 1, further comprising generating an error log, said error log containing a list of test cases each having a test result which failed to match the mapped expected result ~~therefore~~ <u>therefor</u> in a matching test case scenario of the abstraction matrix.

6. (Original) The method of claim 5, wherein said error log further includes the test results of the failing test cases, as well as the mapped expected results from the abstraction matrix for the failing test cases.

7. (Original) The method of claim 1, further comprising creating at least one test results file and at least one abstraction matrix file, and wherein said evaluating comprises automatically reviewing each test result in the at least one test results file by comparing its test case to the at least one test case scenario of the abstraction matrix contained in the at least one abstraction matrix file.

8. (Original) The method of claim 1, wherein the providing comprises creating the abstraction matrix from a functional specification of the software component.

9. (Original) The method of claim 1, wherein the evaluating comprises automatically extracting test statistics, said test statistics including a total number of test cases executed, and at least one of a total number of test cases successfully executed or a total number of test cases unsuccessfully executed.

10. Canceled.

11. Canceled.

12. (Currently Amended) A system for evaluating a software component comprising:

means for deriving a conceptual layering scheme of the software component, the conceptual layering scheme dividing the software component into multiple software layers, the multiple software layers taking into account relationships that exist between software layers;

an abstraction matrix in mathematical abstract form that describes the multiple software layers of the software component and accounts for the relationships that exist between software layers, the abstraction matrix further comprising at least one test case scenario for each software layer and mapped expected results ~~therefore~~ therefor;

means for testing the multiple software layers of the software component using the test ~~cases~~ case scenarios to generate test results; ~~and~~

means for evaluating the test results using the abstraction matrix, the means for evaluating including for each software layer means for automatically comparing a test case employed in the testing of the software layer to the at least one test case scenario of the abstraction matrix therefor, and if a match is found, for automatically comparing the test result for that test case with the mapped expected result ~~therefore~~ for that test case scenario in the abstraction matrix; and

wherein evaluation of the software component is accomplished upon completion of the evaluation of the test results.

13. (Canceled).

14. (Currently Amended) The system of claim 12, further comprising means for deriving at least some test cases for each software layer from the at least one test case scenario of the abstraction matrix for use by the means for testing.

15. (Original) The system of claim 12, wherein the software component comprises multiple states, and wherein the abstraction matrix comprises multiple test case scenarios, each test case scenario being associated with a different state of the software component.

16. (Currently Amended) The system of claim 12, further comprising means for generating an error log, said error log containing a list of test cases each having a test result which failed to match the mapped expected result ~~therefore~~ therefor in a matching test case scenario of the abstraction matrix.

17. (Original) The system of claim 16, wherein said error log further includes test results of the failing test cases, as well as mapped expected results from the abstraction matrix for the failing test cases.

18. (Original) The system of claim 12, further comprising means for creating at least one test results file and at least one abstraction matrix file, and wherein said means for evaluating comprises means for automatically reviewing each test result in the at least one test results file by comparing its test case to the at least one test case scenario of the abstraction matrix contained in the at least one abstraction matrix file.

19. (Original) The system of claim 12, wherein the means for providing comprises means for creating the abstraction matrix from a functional specification of the software component.

20. (Original) The system of claim 12, wherein the means for evaluating comprises means for automatically extracting test statistics, said test statistics including a total number of test cases executed, and at least one of a total number of test cases successfully executed or a total number of test cases unsuccessfully executed.

21. Canceled.

22. Canceled.

23. (Currently Amended) A system for evaluating a software component comprising:

means for deriving a conceptual layering scheme of the software component, the conceptual layering scheme dividing the software component into multiple software layers, the multiple software layers taking into account relationships that exist between software layers;

an abstraction matrix in mathematical abstract form that describes the multiple software layers of the software component and accounts for the relationships that exist between the software layers, the abstraction matrix further comprising at least one test case scenario for each software layer and mapped expected results ~~therefore~~ therefor;

a first computing unit adapted to test the multiple software layers of the software component using the test ~~cases~~ case scenarios to generate test results; ~~and~~

a second computing unit adapted to evaluate the test results using the abstraction matrix, the evaluating including for each software layer automatically comparing a test case employed in the testing of the software layer to the at least one test case scenario of the abstraction matrix therefor, and if a match is found, automatically comparing the test result for that test case with the mapped expected result ~~there~~fore for that test case scenario in the abstraction matrix; and

wherein evaluation of the software component is accomplished upon completion of the evaluation of the test results.

24. (Original) The system of claim 23, wherein the first computing unit and the second computing unit comprise a single computing unit.

25. (Currently Amended) At least one program storage device readable by a machine, tangibly embodying at least one program of instructions executable by the machine to perform a method of evaluating a software component, comprising:

deriving a conceptual layering scheme of the software component, the conceptual layering scheme dividing the software component into multiple software layers, the multiple software layers taking into account relationships that exist between software layers;

providing an abstraction matrix in mathematical abstract form that describes the multiple software layers of the software component and accounts for the relationships that exist between the software layers, the abstraction matrix further comprising at least one test case scenario for each software layer and mapped expected results ~~therefore~~ therefor;

testing the multiple software layers of the software component using the test ~~cases~~ case scenarios to generate test results; ~~and~~

evaluating the test results using the abstraction matrix, the evaluating including for each software layer automatically comparing a test case employed in the testing of the software layer to the at least one test case scenario of the abstraction matrix therefor, and if a match is found, automatically comparing the test result for that test case with the mapped expected result ~~therefore~~ for that test case scenario in the abstraction matrix; and

wherein evaluation of the software component is accomplished upon completion of the evaluation of the test results.

26. (Canceled).

27. (Currently Amended) The at least one program storage device of claim 25, further comprising, prior to said testing, deriving at least some test cases for each layer from the at least one test case scenario of the abstraction matrix.

28. (Original) The at least one program storage device of claim 25, wherein the software component comprises multiple states, and wherein the abstraction matrix comprises multiple test case scenarios, each test case scenario being associated with a different state of the software component.

29. (Currently Amended) The at least one program storage device of claim 25, further comprising generating an error log. said error log containing a list of test cases each having a test result which failed to match the mapped expected result ~~therefore~~ therefor in a matching test case scenario of the abstraction matrix.

30. (Original) The at least one program storage device of claim 29, wherein said error log further includes test results of the failing test cases, as well as mapped expected results from the abstraction matrix for the failing test cases.

31. (Original) The at least one program storage device of claim 25, further comprising creating at least one test results file and at least one abstraction matrix file, and wherein said evaluating comprises automatically reviewing each test result in the at least one test results file by comparing its test case to the at least one test case scenario of the abstraction matrix contained in the at least one abstraction matrix file.

32. (Original) The at least one program storage device of claim 25, wherein the providing comprises creating the abstraction matrix from a functional specification of the software component.

33. (Original) The at least one program storage device of claim 25, wherein the evaluating comprises automatically extracting test statistics, said test statistics including a total number of test cases executed, and at least one of a total number of test cases successfully executed or a total number of test cases unsuccessfully executed.

34. (Canceled).

35. (Canceled).

36. (New) The method of claim 1, wherein the providing further includes parsing the abstraction matrix to automatically generate and factor out doable test cases and mapped expected results therefor, and separating the test cases based on the software layers of the software component, and associating data structures with the separated test cases of the software layers, the data structures allowing the test cases of the various software layers to be uncorrelated, and wherein the abstraction matrix comprises an abstraction file for each software layer, and the parsing comprises generating a mapped expected result for each line of each abstraction file based on associated state information.

37. (New) The method of claim 36, wherein the separating comprises eliminating dependencies between at least one input of a software layer based on other inputs of other software layers of the software component, thereby uncorrelating test cases of the various software layers.

38. (New) The method of claim 37, wherein the data structures for at least some software layers mimic information passed between layers during normal operation of the software component.

39. (New) The method of claim 38, wherein the software component comprises a cluster operating system component.

* * * * *